VIPER

March - April 1984
Volume 5 Number 6

Journal of the VIP Hobby Computer Assn.

The VIPER was founded by ARESCO, Inc. in June 1978

****

PIPS for VIPS IV - Part 6

Well, we have finally finished the serialization of Tom Swan's PIPS for VIPS IV. This issue deals mostly with Graphics in machine language, and has the code for the subroutines not yet covered. This means that the documentation for PIPS IV is now complete. By placing the past year's issues in a binder, you will have a complete copy of PIPS IV. You will notice that I've tried to keep un-PIPS-like items on the cover pages, so that you can even eliminate them from your collection. I'd like to thank Tom Swan for this monumental contribution to VIPER and I'm glad that it was possible to get the final PIPS published.

Cassettes of PIPS IV are still available and the cost for the ten programs on the tape is $5. You may send your check to VIPHCA at the usual address, 32 Ainsworth Avenue, East Brunswick, NJ 08816. Each program on the tape is complete, with the appropriate CHIP-8 interpreter, where necessary. The program titles are:

1. Cos Melodeon
2. Holiday tree
3. Attack of Micromen
4. Line up
5. Sweeper
6. Move on
7. Box
8. Shape maker
9. Graphics #1
10. Graphics #2

VIP Hobby Computer Association
32 Ainsworth Avenue
East Brunswick, NJ 08816

—:— News and Views —:—

Well Gang, we have completed yet another year of VIPER! My thanks once again to Tom Swan for PIPS IV. The question now, however, is, "Where do we go from here?" I've been worrying over this one for quite a while, and I'm sure most of you have had some concern for VIPHCA's fate after RCA discontinued the manufacture of VIP related items.

I've had notes and letters from a number of you exorting me to "keep it going" - "there's no other publication for the 1802," etc. But we are going to have to do some hard thinking about this. Each year that it comes time for you out there to renew membership in VIPHCA (meaning, send in your dues) there are some members who do not renew. And this is understandable, since some folks lose interest, or move, or get newer and better computers, and so on. This decrease in membership means not only fewer members, but a geater burden on the dues of each member. We could raise our dues, but that would probably cause even more members to drop out, making the situation worse, not better. It's tough to keep an organization growing when there is no way for new people to get the equipment to join in.

I've been able to effect some economies in the production of VIPER this year, but even so, it is still costing more per issue to publish than two years ago, when we had more members. But it would be a shame to let VIPER fall by the wayside. Perhaps the answer is to expand the sphere of our interests and have material about other low-cost computers. But even there, there's been attrition in the ranks of the manufacturers. Timex has also thrown in the towel. Rumor has it that Radio Shack will discontinue its MC-10 after less than a year on the market. The VIC-20 is still available, but it's days may be numbered. Don't get me wrong, I'm not bitter about this at all--there is a time for everything and we must allow for "progress". You can buy a lot of computer today for what we paid for our VIP and ELF machines.

Please let me know what you would like VIPHCA to do. Since this is the time for you to send in your $12 dues, send along a note with your thoughts on where we should go from here. Even if you decide not to renew, I'd still like to hear from you!

SHAPE MAKER

I wrote SHAPE MAKER for several reasons. First, it is a work horse utility that will help make the creation of shape tables easier. Second, I mentioned earlier that register restrictions weren't as bad as they may seem with the Graphics Subs packages. SHAPE MAKER doesn't even use R8, and it is quite an involved program so it is running proof of my conjecture. Third, and this may be the most important, I sense that many of you with little or no 1802 machine language experience may have trouble getting started with using the graphics subs. In particular, I haven't said anything about how to get the hexpad working, usually an important feature of any VIP programming and essential for games. SHAPE MAKER contains many subroutines which are general enough for including in your own programs and contains a lot of answers to potential programming questions for those of you who want to put on your shiny, black FBI shoes and have a look inside the program.

What's it do? You have probably used one of the many CHIP-8 drawing programs to "doodle" on your TV screen. Maybe you have written one. SHAPE MAKER is similar to those programs but only because it too gives you a way to draw pictures on the display. However, the use and purpose of SHAPE MAKER is completely different from any of the doodlers I've seen for the VIP.

Here is a list of the hex key functions available in SHAPE MAKER:

HEX KEY	FUNCTION
0	No function
1	Move cursor 315 degrees (NW)
2	Move cursor 0 degrees (N)
3	Move cursor 45 degrees (NE)
4	Move cursor 270 degrees (W)
5	No function
6	Move cursor 90 degrees (E)
7	Move cursor 225 degrees (SW)
8	Move cursor 180 degrees (S)
9	Move cursor 135 degrees (SE)
A	Put current shape on hold
B	Recover shape on hold
C	Draw the current shape
D	Draw line to the cursor
E	Erase last line drawn
F	Start over

USING THE PROGRAM

Load five pages from tape, or if you are loading by hand, copy the listing exactly as shown into the addresses $0300 to $04FF and "mate" with a copy of Graphics Subs Package #1 in locations $0000 to $01FF.

When you flip the run switch up, the screen will be blank and the program is expecting you to enter a starting X,Y coordinate which will be displayed in the upper left corner of the display screen. You must enter leading zeros, so to enter the X,Y coordinates (1,F) you would punch in (not too hard) 01 then 0F.

The screen will clear when you enter the fourth digit and a cursor will be blinking at your X,Y coordinates. If you happen to enter coordinates out of the display range ($0 \leq X \leq \$3F$ and $0 \leq Y \leq \$7F$) the program automatically adjusts the bad input (rather than

rejecting it) and puts the cursor on the display.

There is no escape from this initializing mode and you must always enter a starting X,Y when the program is expecting it in order to begin drawing shapes.

THE CURSOR

Use the normal VIP hex key format to move the cursor in eight directions corresponding to the keys 1,2,3,4,6,7,8,9. A little practice with these keys will reveal their functions. The cursor cannot be sent past the edges of the display; there is no wrap around. This makes finding the corners and edges of the display window easy. Notice that when using a diagonal key (#9 for example) the cursor continues to go straight down when running into the right wall until it reaches the bottom right corner. The other diagonal keys work similarly.

The cursor is "non-destructive" meaning you can pass through shapes already on display without disturbing their integrity. (Honest!)

DRAWING A LINE

After entering a starting X,Y coordinate move the cursor to the point <u>to</u> which you want to draw a line. Pressing Key D will connect the starting X,Y (now invisible) and the cursor. Moving the cursor away from that point to another and pressing Key D will

cause a second line to be drawn, etc. Rather than leaving a trail behind as in other doodle programs, the cursor in SHAPE MAKER is only a position finder. The subroutine LINE in the Graphics Subs Package is called to actually draw a line to the present cursor position. Up to 126 points may be connected before the Shape Table buffer overflows. (More on that in a moment.)

ERASING A LINE

Whenever you wish, you may press Key E to erase the last line which was drawn on the screen. The cursor may be anywhere. Of course if there are no lines there to erase, pushing Key E will have no effect. After erasing a line, the cursor will always be positioned at the end of the last line displayed -- you could say at the "head" of the current shape -- and you may continue drawing from that point.

STARTING OVER

Pressing Key F takes you back to the beginning of the program to wait for new starting X,Y coordinates just as when you first flip up the run switch. The screen is cleared and the current shape is wiped from memory.

SEEING YOUR SHAPE

Key C draws the current shape. Since you are already seeing that shape, however, pressing Key C will simply redraw what you are seeing. This may seem a bit useless, but because the program is controlled by a jump table, implementing this function was as simple as entering two bytes into that table.

There are a few occasions when Key C will be a help. When you want to see how fast a shape will be drawn, pressing Key C shows you what drawing the shape will look like perhaps in a program of your own design. It also serves as a confirmation that the shape table is being properly recorded. And when you want to return the cursor to the head of the shape, Key C gets you there.

Those are the four main functions of SHAPE MAKER forming a control center of hex keys along the right side of the VIP hex pad. There's more, but first let's look at what SHAPE MAKER is doing with the lines you have drawn.

THE SHAPE TABLE BUFFER

Each time you draw a line to the cursor, SHAPE MAKER inserts the X and Y coordinates of the cursor into an area of memory from $0700 to $07FF. This page is called the SHAPE TABLE buffer and may contain up to 127 X,Y points including the starting X,Y plus

an ending $FFFF two byte stop indicator. (In other words, 126 lines is the maximum that may be drawn from the SHAPE TABLE buffer.)

Erasing a line by using Key E removes the last X,Y pair from the shape table, replacing those coordinates with an $FFFF stop indicator. Drawing a line inserts X and Y and an $FFFF byte pair into the SHAPE TABLE. On start up, the entire buffer is set to $FF bytes as an extra precaution against drawing runaway shapes. This also makes finding the end of a shape table a little easier when doing several shapes for the MERGE subroutine. Just look for all the $FF's.

After drawing your shape, you may save it on tape by recording one page beginning at $0700. That shape may then be used in the other programs in this book that work with shape tables.

HOLDING AND RESTORING SHAPE TABLES

Pressing Key A puts the current shape on hold, clears the screen and asks for new starting X,Y coordinates, the same as pressing Key F. However, before wiping the current shape from memory, Key A's routine copies the SHAPE TABLE buffer from $0700 to $0800 thus preserving your shape.

You may then draw new shapes without disturbing the old one. In the program's present version, there is no way to show two shapes at one time on the display screen.

Pressing Key B brings the shape on hold back into the SHAPE

TABLE buffer so you may continue working on it. The shape is still on hold at the same time and may be brought into the display area at any time, as many times as you wish.

There may only be one shape on hold at a time. Putting a new shape on hold replaces whatever shape was previously on hold.

Restoring a shape from the hold buffer when there is no shape on hold will produce strange effects and may cause the cursor to be positioned out of its display range (though it will be visible). In that case Key F gets you out of trouble.

HINTS ON USING

The more I use this program, the more I discover things that may be done with it. Here are some suggestions:

To save a shape on a cassette for working on later, record the SHAPE TABLE buffer at $0700 as said earlier. To bring that shape back, load one page at $0800. This puts the old shape from the cassette into the hold buffer. Then after entering a starting X,Y pair, pressing Key B will bring the shape into the SHAPE TABLE buffer for viewing or changing.

You may enter X,Y coordinates into the HOLD buffer using the write mode of your VIP too. Simply start at $0800, enter a starting X,Y pair, then as many X,Y pairs as you want up to and including address $08FD. Be sure to end a hand loaded SHAPE TABLE with $FFFF, or Key B will cause it to be bent out of shape! See the chapter on the SHAPE sub in the Graphics Subs Packages section

for the SHAPE TABLE format. Be sure to use X,Y coordinates within the proper ranges.

Because pressing Key F is potentially dangerous -- it wipes the current SHAPE TABLE buffer clean -- I suggest you frequently use Key A to preserve your work up to a certain point in the HOLD buffer. With a shape on display, first press Key A. Enter any starting coordinates (0000 -- FFFF -- it doesn't matter here) then press Key B. Now you may keep right on working with your shape. If you happen to press Key F accidentally, you can always recover your shape as it was last saved rather than having to start over.

I thought I didn't program a clear-screen-home-cursor function, but I find now that I did! Enter a starting X,Y of 00,00 then press Key A when the cursor appears. You have to re-enter 00,00 and may then begin drawing shapes. What you have done is to create an empty SHAPE TABLE in the HOLD buffer. Everytime you press Key B, the "non shape" is brought into view thus clearing the screen and returning the cursor to its "home." You may enter any starting X,Y coordinates other than 0,0 -- maybe the center of the screen would be useful -- and follow the above procedure. Pressing Key B will "home" the cursor to your selected coordinates -- a programmable home function!

To see your starting point on the screen, press Key D once <u>before</u> moving the cursor the first time. Now when you move the cursor away from the starting X,Y position, that position will show up as a stationary white dot. Of course, the dot's

coordinates have been entered into the shape table (actually twice as the first line drawn is just from and to the same point). This may or may not be important depending on how you plan to use the SHAPE TABLE.

SOME CAUTIONS

Pressing Key D (draw line) several times in succession, while not having a visible effect, will cause the current X,Y coordinates of the cursor to be repeatedly entered into the SHAPE TABLE. Though these unnecessary points will not affect the drawing of your shape (except to slow it down a little) the extra points will appear to disable Key E (erase line). If you press Key E and nothing appears to happen, this is the probably cause. You have to press Key E once for each of the extra X,Y pairs in the SHAPE TABLE until lines will begin to be erased again. If Key E seems to stop working, just keep pressing it. Eventually it will come out of it.

When you have entered the maximum number of lines, SHAPE MAKER will refuse to draw any more. If Key D (draw line) has stopped functioning, then the SHAPE TABLE is full. Full SHAPE buffers may be placed on hold and recovered and worked on exactly as others except that the $FFFF stop indicator will not be sent to the HOLD buffer. This causes no problems when using SHAPE MAKER as the SHAPE TABLE buffer is always ended with $FFFF. However,

if you record a full SHAPE TABLE from the HOLD buffer at $0800, it will not contain the $FFFF stop indicator. For this reason, it is best to always record from the SHAPE TABLE buffer at $0700 and load into the HOLD buffer at $0800. This will guarantee that any size SHAPE TABLE will be properly terminated.

All lines are drawn with $012D in the POINT sub set to $F1, the 1802 logical OR instruction. However, to get the cursor to blink and move, this location is changed from $F1 to $F3 (logical XOR) then back again. In the listing, you will see the subroutine ORXOR which does the job of changing that memory location. This is some of the self-modifying code you may have heard me be cautious of in the past. Be careful. While seemingly harmless, the ORXOR sub is a loaded weapon using a breed of insect commonly referred to unromantically as just plain "BUG!" If you reset the VIP when $012D is one way, rerunning SHAPE MAKER would then cause that byte to possibly start out with the wrong value. In other words, we want to go flip-flop-flip (OR-XOR-OR) but if it starts out with $012D=$F3=XOR, the program will go flop-flip-flop (XOR-OR-XOR) and won't work at all!

This is easily fixed by initializing $012D to $F1=OR early in the program. Now, when resetting the computer, it doesn't matter which way $012D is set, it will be properly reset on the next run.

But don't breathe too easily. Let's say you want to make a

copy of SHAPE MAKER*. First you run the program, then record it somewhere. Fine for SHAPE MAKER. Then one day next year you decide to use the Graphics Subs Package from the first two pages of SHAPE MAKER but it doesn't work as it should! Because you can never be sure which way $012D is set after running SHAPE MAKER, it is best never to use its Graphics Subs Package for other programs. See what I mean about self-modifying code?

GO AHEAD -- BUTCHER IT

The listing for SHAPE MAKER is well documented but there are several non-graphics subroutines you may want for your own programs. The ones that get the hexpad working are particularly important.

It is not possible for you to use the hexpad sub included in your ROM VIP operating system with the Graphics Subs. That routine requires the interrupt timer R8 to be decremented every 1/60 seconds and the Graphics Subs Packages' interrupts do not do this to R8.

The subs GETKEY and CHEKEY (CHECK-KEY) at $0400 and $0419 give you similar hexpad operations that you have when programming in CHIP-8. GETKEY will wait until any key is pressed, then return the hex digit in the high part of register RF (RF.1) corresponding to that key. It works similarly to the ROM routine except that

---

it uses its own timing loops to sound the tone and debounce the key press. $\overline{EF3}$, the 1802 flagline, is true only if the low four bits of the byte sent to the hexpad circuit matches that produced when pressing on one of the keys. At least that's a reasonable software explanation of how it works. See your manual if you want to know more about the electrical connections.

To use GETKEY, simply call the subroutine. Afterwards, RF.1 contains the hex digit of the key pressed.

CHEKEY works a little differently. It checks to see if a particular key is being pressed but does not wait for a keypress signal. In this sense it is a dynamic routine because it may be used continuously in a program loop without holding up the works.

To use CHEKEY, first preset RF.1 equal to the hex key value you want to check. The first four bits are of no importance in this value so that A1 or 61 or 01 would all work to check if Key 1 is being pressed.

When the subroutine returns to your main program, RF.0 will contain either the value 00 or 01. If it is 00 then that key is not being pressed. If it is 01 then the key in question is feeling the pressure.

For an example of one way to use CHEKEY, study the instructions from $0312 to $031F in the program's main loop. RF.1 is cycled through the values $0F to $00 by one in that order. As long as no key is being pressed, then the program will continue to loop around, decrementing RF.1 on each pass, and blinking the cursor

every sixteen loops. Only if a key is being pressed will the branch at 031F not be taken. If any key is being pressed, then control will "fall through" that point continuing on to the instructions at $0321-$032B. When that happens, the value of the hex key pressed is in RF.1.

At that point a test is made to see if RF.1 is less than $0A. If it is, then the branch at $0324 is taken, going directly to the point in the program that calls the various function subroutines. However, if the key pressed is A or greater, the branch at $0326 holds control there until the key is released making $\overline{EF3}$ not true. Then the function will be performed. This is important as it allows the keyboard to be split up into those keys which automatically cause something to happen as long as they are being pressed, and those that will select a function only one time when they are pressed and released. Obviously, you wouldn't want to press a key each time you want the cursor to move, and erasing the last ten lines drawn before you can get your finger off the button wouldn't make life easy.

Though your program may use a different register for holding X,Y coordinates, the MOVE subroutines should be helpful to you. These are located from $0397 through $03D0 and form an eight subroutine block that may be used to change X and Y in any of eight directions. Notice that the diagonal subs simply call the horizontal and vertical subs in order to move the cursor. Also notice that there are redundancies which could have been replaced

by short branches. I much prefer having each sub do its own job rather than jumping around just for the purpose of saving a few program bytes.

The same philosophy, by the way, has been followed throughout the program to make hacking it to pieces that much more pleasant. In particular, one optimization technique that would result in an increased running speed has been avoided. The technique suggests looking for subroutines that end with a call to another subroutine before executing a return. In that case, a branch to the subroutine would work as well with that routine executing the final return.

For example, look at the MOVE sub at $03A5. The last CALL may be replaced with a short branch to $03CA for a two byte savings and a speed increase because the SCRT CALL and RETN subs are executed once less each. Even juicier is the prospect of simply removing the CALL and RETN instructions at 03AF and 03B2. Since that sub is just going to CALL the next one in line, it could be left to run on through directly to the next routine without any ill effects.

This is an important optimization technique and I use it and others frequently. But linking routines together in this way must be done only to programs that are completely debugged and hopefully in their final forms. Otherwise, moving or altering one section of the program could have disastrous effects on who knows how many others. In SHAPE MAKER I purposely made every subroutine

hold its own. Not only is the program more easily wrenched apart (hope you got your wrenches handy), but I think each routine is easier to read and understand this way.

USING THE ROM HEX DISPLAY

The subroutine at $0426 may be used to display hex digits. Bit patterns for these digits are kept in the VIP operating system ROM and accessed by the DSPHEX sub. Place the digit you want to display in RF.1, set RD to some address within the display page and call DSPHEX. Note that RD is an address not a coordinate where you want the top of the hex digit to appear. Double precision arithmetic is used so that digits may be displayed in high resolution though except for that, the ROM routine would work just as well. (See VIPER Vol 1, Issue 3 for a documented listing of the VIP operating system routines.)

DOUBLD is really just a combination of GETKEY and DSPDIG permitting keyboard entry of two digits, returning those digits as a single eight-bit byte value in RF.1. Again, RD must be properly set before calling DOUBLD.

JUMP TABLES WITH SCRT

Using the SCRT for subroutines complicates one of the nice features about 1802 machine language programming. Usually, any register may be used as the program counter. Subroutine addresses

may be kept in a table and simply loaded into a register. Making that register the program counter serves as the fastest means in microcomputing of calling a subroutine.

JUMPER is a little program that allows a similar technique in SCRT programs, though it lacks the speed of the other method.

At first this routine may seem a little strange. The first line of the sub actually calls its own following instruction! What this does is to push the return address of the main calling routine onto the stack -- there will never be a return to location $3D4. At this point R6 is equal to $03D4 to where if a return were executed,.the program would proceed by setting R6 to the address of a sub kept in the jump table (JMPTAB), the next execution of a RETN will cause that sub to begin running. Actually, the RETN is being used to CALL subs which presumably end with their own RETN instruction. That RETN passes control back to the routine that originally called JUMPER using the address from the stack put there with the call at $03D1. Most important, JUMPER is capable of functioning in Read Only Memory. If this seems confusing, study the CALL and RETN routines to see how they work. JUMPER is a little madness added to those routine's methods.

MODIFICATIONS -- A WALK THROUGH

Two keys on the hex pad have not been used. You may write your own routines and insert their addresses into the Jump Table

to enable new functions. End your subroutines with a $D5 RETN instruction.

You may easily change what keys do what by switching around the subroutine addresses in the Jump Table. For example to make Key C do what Key D does and vice versa, just enter $046A at $03F8 and $045E at $03FA. Remember that Keys 0 through 9 are instantly active while Keys A through C must be pressed and released to work.

For an example of how to add your own routines, first load in SHAPE MAKER, then follow along while we walk through the necessary steps.

It may be helpful to know at what address the SHAPE TABLE extends to so that MERGE shapes may be constructed to be all the same length. Wouldn't it be nice to have SHAPE MAKER display that address rather than having to look through each SHAPE TABLE for the $FFFF stop indicator and then adjust each table to size by hand?

First decide on which key you want to select this function. I'll pick Key 0. Though it is one of the instantly active keys, there's an easy way around that limitation.

Next, what are the things the routine needs to accomplish? It is a good idea to just write all the steps down rather than try to program or write a flowchart at this point. In fact, if you state the function clearly, flowcharts will often not be needed.

This is my list of steps:

1) Clear the display
2) Show the last address used in the SHAPE TABLE
3) Erase the address
4) Return the shape to the display

Thinking about each separate step in this way helps simplify writing the final routine. Numbers 1 and 3 are easy. The Graphics Subs Package CLEAR sub will handle both of these with a single call. Number 4 is also uncomplicated. Remember Key C? It draws the current shape. Turning to that subroutine at $045E, we see that all input requirements have already been met. R7.1 is set before the main loop begins, and we know there is a shape table in the buffer. So again, number 4 may easily be programmed with a simple call to $045E.

On careful examination of the KEYC sub, notice that before drawing the shape, the screen is cleared. Bells and whistles shoulds be going off. Look again at the list of steps above. Number 3 is not really needed -- the CALL to KEYC will take care of erasing the address.

Now let's see. First we clear the screen, show the address then return the old shape. Oops! That won't give you much time to write the address down, maybe not even to see it at all! Better make #3 above read "Wait for any keypress." Once again that is pretty easy. Just call GETKEY and the program will wait till any key is pressed.

That leaves number 2 for which there are no subs to do the whole job. We want to display a double hex digit number, probably using the subroutine DSPHEX. At this point you may want to write down the individual steps for number 2 -- set RD, display one digit, display other digit, etc. -- and you're ready to program.

Key 0 is the only snag. We want to have it act like Keys A through F or the routine will be over and done before you can get your finger off the key. Because EF3 will be true if a key is still being pressed on entry to any sub, it can be tested to be sure the key will be released before going on. This is best done at the entry to the sub.

Now pick an unused area of memory, let's start at $0330, and using the list of steps, program each one. Here's what I came up with. I called it the ADDRS sub.

```
0330  36 30      ADDRS: B3   ADDRS  ;Wait here till key is released
  32  D4 01 30           CALL CLEAR  ;Do number one.  Clear display
  35  9B                 GHI  RB     ;Set RD to the address in
  36  BD                 PHI  RD     ;  the display where digits are to
  37  F8 F0              LDI  $F0    ;  go.  See INPUT to DSPHEX
  39  AD                 PLO  RD
  3A  87                 GLO  R7     ;Get address of SHAPE TABLE
  3B  F6                 SHR         ;  end from R7.0.  We want to
  3C  F6                 SHR         ;  show the first digit, so
  3D  F6                 SHR         ;  shift it to the right
  3E  F6                 SHR
  3F  BF                 PHI  RF     ;Put in RF.1 to pass to sub
0340  D4 04 26           CALL DSPHEX ;Do sub to display the digit
  43  87                 GLO  R7     ;Get address again and mask
  44  FA 0F              ANI  $0F    ;  with AND for second digit
  46  BF                 PHI  RF     ;Put in RF.1 to pass to sub
  47  D4 04 26           CALL DSPHEX ;Do sub to display the digit
  4A  D4 04 00           CALL GETKEY ;Wait for any keypress
  4D  D4 04 5E           CALL KEYC   ;Show the shape again
  50  D5                 RETN        ;End -- return to main loop
```

Now that the routine has been written and entered into memory, it has to be enabled. To do this, the address of the sub must be entered into the Jump Table. Turning to JMPTAB at $03E0, look for the reserved space for KEY 0. In this case, that happens to be at

$03E0 so enter $0330 at that address and you're done. (If it works.) Want Key 5 instead? Then enter $0330 at $03EA instead. It's that simple.

I haven't gone through all the mental processes involved. You need to be familiar with the register use, etc. After all, I knew that R7.0 addresses the end of a shape table at all times and splitting a byte into its four-bit halves is no new trick to me. But that's generally how you may want to go about adding functions to SHAPE MAKER. Good luck!

SUGGESTIONS

When you do add a function, send it in to the VIPER so all of us may share your discovery. Here's a few ideas to get you started.

1) New HOLD function that doesn't go back to the initializing routine. Faster than Key A for saving partially created SHAPES.
2) Display shape table in hex. Allow entry of bytes from the hex pad. Like a CHIP-8 editor.
3) Combine hold and current shape. Restores the HOLD buffer and overdraws shape without erasing the one already on display.

For more ambitious programming sessions:

1) Let HOLD Key A select a bank of memory pages into which you may put several SHAPES on hold. Allow restoring any

one of these SHAPES from any of the various HOLD buffers.

2) Merge two HOLD buffers together. Requires extensive relocating of MERGE sub and probably a new MERGE controller.

3) Rearrange all SHAPE buffers into one long string of SHAPES ending with an extra $FFFF stop indicator and ready for MERGing.

4) Reposition an entire SHAPE by a specified amount. Adds or subtracts a quantity from all X,Y's in a table keeping the results within display limits.

5) Add rotation, scale, stretch features using the same idea. Formulas available in most computer magazines.

SHAPE MAKER

REGISTER ASSIGNMENT

R0 - DMA Pointer
R1 - Interrupt routine program counter
R2 - Stack Pointer
R3 - Normal program counter
R4 - CALL routine program counter
R5 - RETN routine program counter
R6 - Pointer to return and arguments
R7 - Addresses SHAPE TABLE buffer at $0700-$07FF
R8 - Not used
R9 - Used by Graphics Subs
RA - " " "
RB - RB.1=display page starting address
RC - Cursor X,Y (RC.0=X, RC.1=Y)
RD - Destination address for hex digits
RE - Utility
RF - Utility - RF.1 used in hexpad routines

Also see Graphics Subs Register Assignment and Memory Map for additional details.

MEMORY MAP (4K)

0000 - 01FF -- Graphics Subs Package #1 (see ORXOR warning)
0200 - 02FF -- Stack
0300 - 04FF -- SHAPE MAKER program -- 032D-0372 available
0500 - 06FF -- Available for expansion
0700 - 07FF -- SHAPE TABLE buffer
0800 - 08FF -- HOLD buffer
0900 - 0BFF -- Available for expansion
0C00 - 0FFF -- Display refresh

SYMBOL TABLE

*BEGIN - 0300	D0 - 03B3	DSPHEX - 0426
MOVMEM - 0373	D90 - 03BA	DOUBLD - 0447
HOLD - 0385	D180 - 03C1	KEYC - 045E
RECVR - 038E	D270 - 03CA	KEYD - 046A
D45 - 0397	JUMPER - 03D1	KEYE - 0489
D135 - 039E	JMPTAB - 03E0	ORXOR - 04A7
D225 - 03A5	GETKEY - 0400	CURSOR - 04B2
D315 - 03AC	CHEKEY - 0419	TIMER - 04CD
		INIT - 04D5

*SHAPE MAKER runs as a non returning "subroutine" of the Graphics Subs. Except for BEGIN, all of the above routines may be called as individual subroutines from your own added functions.

SHAPE MAKER

```
0300  F8 07     BEGIN: LDI  $07     ;Initialize R7.1=07=start
  02  B7               PHI  R7      ;  address of shape table
  03  F8 01            LDI  $01     ;Set RF to address the
  05  BF               PHI  RF      ;  OR-XOR instruction in
  06  F8 2D            LDI  $2D     ;  the POINT sub
  08  AF               PLO  RF
  09  F8 F1            LDI  $F1     ;Set that instruction to OR ($F1)
  0B  5F               STR  RF      ;Needed possibly on using reset
                                    ;  switch
  0C  D4 04 D5         CALL INIT    ;Initialize everything

                ;Begin Main Loop

030F  D4 04 B2    1H: CALL CURSOR  ;Blink cursor on/off
  12  F8 10            LDI  $10     ;Preset RF.1=$10 to check
  14  BF               PHI  RF      ;  Keys 0 thru F
  15  9F          2H: GHI  RF      ;Subtract 1 from value
  16  FF 01            SMI  $1      ;  in RF
  18  3B 0F            BM   1B      ;If minus (RF.1 was 00),take branch
  1A  BF               PHI  RF      ;Else put in RF.1
  1B  D4 04 19         CALL CHEKEY  ;See if that key is depressed *
  1E  8F               GLO  RF      ;Test flag returned from CHEKEY
  1F  32 15            BZ   2B      ;If=0 then loop.  Key not pressed
  21  9F               GHI  RF      ;Test if RF.1 (key pressed)
  22  FF 0A            SMI  $A      ;  is greater than $09
  24  3B 28            BM   4F      ;If not, then go to function
  26  36 26       3H: B3   3H      ;Else wait till key is released
  28  D4 03 D1    4H: CALL JUMPER  ;Then go jump to function
  2B  30 0F            BR   1B      ;Loop all over again

                ;End Main Loop

032D to 0372 -- Available
```

*Poor key. Cheer up, you could be in a Radio Shack computer.

MOVE MEMORY SUB

```
                ;Input:    Page addresses as two byte OMON
                  parameter following CALL
                ;Output:   Page OM transferred to page ON less
                  last two bytes
                ;Called by:HOLD  RECVR
                ;Changes:  R6(return address), RE, RF

0373  46        MOVMEM: LDA  R6      ;Pick up address parameters
  74  BE                PHI  RE      ;  and place in RE (from page)
  75  46                LDA  R6      ;  and in RF (to page)
  76  BF                PHI  RF
  77  F8 FD             LDI  $FD     ;Then set low byte =$FD
  79  AE                PLO  RE      ;  to never replace the last
  7A  AF                PLO  RF      ;  two ($FF) bytes in table
  7B  EF                SEX  F       ;X=F for using STXD instruction
  7C  0E           1H:  LDN  RE      ;Move memory from M(R(E))
  7D  73                STXD         ;  to M(R(F)) incrementing
  7E  2E                DEC  RE      ;  both pointers
  7F  8E                GLO  RE
  80  3A 7C             BNZ  1B
  82  0E                LDN  RE      ;Take care of the last
  83  5F                STR  RF      ;  byte here
  84  D5                RETN         ;Return from subroutine
```

HOLD/RECOVER SHAPES

```
                ;Input:    None for either sub
                ;Output:   Appropriate page transfer
                ;Called by:Main Loop
                ;Calls:    MOVMEM  INIT  KEYC
                ;Changes:  No registers directly

            PUT SHAPE ON HOLD (M($700) to M($800))

0385  D4 03 73    HOLD:  CALL MOVMEM ;Do memory move sub
  88  07 08               .WRD=$07,$08;Pass from/to address parameters
  8A  D4 04 D5           CALL INIT   ;Re-initialize
  8D  D5                 RETN        ;Return

                RECOVER SHAPE ON HOLD

  8E  D4 03 73  RECVR:  CALL MOVMEM ;Do memory move sub
  91  08 07               .WRD=$08,$07;Pass from/to address parameters
  93  D4 04 5E           CALL KEYC   ;Display shape & set R7, RC
  96  D5                 RETN        ;Return
```

MOVE CURSOR SUBS

```
                    ;Input (All moves): RC=cursor in proper range
                    ;Output:   RC adjusted accordingly limited to
                       screen edges
                    ;Called by:Main Loop (straights called by diagonals)
                    ;Calls:    (diagonals call straights)
                    ;Changes:  RC probably
```

MOVE 45 DEGREES

```
0397  D4 03 B3    D45: CALL D0      ;Move 0 and 90 degrees
 9A   D4 03 BA         CALL D90     ;  equals 45
 9D   D5               RETN         ;Return
```

MOVE 135 DEGREES

```
039E  D4 03 BA   D135: CALL D90     ;Move 90 and 180 degrees
 A1   D4 03 C1         CALL D180    ;  equals 135
 A4   D5               RETN         ;Return
```

MOVE 225 DEGREES

```
03A5  D4 03 C1   D225: CALL D180    ;Move 180 and 270 degrees
 A8   D4 03 CA         CALL D270    ;  equals 225
 AB   D5               RETN         ;Return
```

MOVE 315 DEGREES

```
03AC  D4 03 CA   D315: CALL D270    ;Move 270 and 0 degrees
 AF   D4 03 B3         CALL D0      ;  equals 315
 B2   D5               RETN         ;Return
```

MOVE 0 DEGREES

```
03B3  9C          D0: GHI  RC       ;Get Y coordinate
 B4   32 B9           BZ   1F       ;If already=00, branch to exit
 B6   FF 01           SMI  $1       ;Else subtract 1
 B8   BC              PHI  RC       ;Put new Y back
 B9   D5          1H: RETN          ;Return
```

MOVE 90 DEGREES

```
03BA  8C          D90: GLO  RC      ;Get X coordinate
 BB   FB 3F           XRI  $3F      ;If already = $3F
 BD   32 C0           BZ   1F       ;  then branch to exit
 BF   1C              INC  RC       ;Else add 1 to X (RC.0)
 C0   D5          1H: RETN          ;Return
```

MOVE 180 DEGREES

```
03C1  9C          D80: GHI  RC      ;Get Y coordinate
  C2  FC 01            ADI  $1      ;Add 1
  C4  FE               SHL          ;Test if that makes it > $7F
  C5  33 C9            BDF  1F      ;If so, branch to exit
  C7  F6               SHR          ;Else restore the value
  C8  BC               PHI  RC      ;Put new Y back
  C9  D5           1H: RETN         ;Return
```

MOVE 270 DEGREES

```
03CA  8C         D270: GLO  RC      ;Get X coordinate
  CB  32 D0            BZ   1F      ;If already=00, branch to exit
  CD  FF 01            SMI  $1      ;Else subtract 1
  CF  AC               PLO  RC      ;Put new X back
  D0  D5           1H: RETN         ;Return
```

JUMPER

```
                ;Input:    RF.1=a hex key digit in the form ON
                ;Output:   Subroutine from JMPTAB is called
                ;Called by:Main Loop
                ;Calls:    Itself and the subroutine indirectly
                ;Changes:  R6, RF

03D1  D4 03 D4 JUMPER: CALL JUMP2  ;Push the return address onto stack
  D4  9F         JUMP2: GHI  RF    ;Get passed hex digit
  D5  FE                SHL        ;Multiply by 2
  D6  FC E0             ADI $JMPTAB.0;Add base address of table
  D8  AF                PLO  RF    ;Put in RF.0
  D9  93                GHI  R3    ;Set RF.1 = page address
  DA  BF                PHI  RF    ;  of table = PC.1
  DB  4F                LDA  RF    ;Get address from table
  DC  B6                PHI  R6    ;  and place in R6 holding
  DD  0F                LDN  RF    ;  the "return" jump address
  DE  A6                PLO  R6
  DF  D5                RETN       ;"Return" to call subroutine
```

JUMP TABLE

```
03E0  03DF     JMPTAB ;KEY 0 - No function, return
  E2  03AC            ;KEY 1 - Move diagonally 315 degrees
  E4  03B3            ;KEY 2 - Move straight 0 degrees
  E6  0397            ;KEY 3 - Move disgonally 45 degrees
```

```
03E8  03CA              ;KEY 4 - Move straight 270 degrees
  EA  03DF              ;KEY 5 - No function, return
  EC  03BA              ;KEY 6 - Move straight 90 degrees
  EE  03A5              ;KEY 7 - Move diagonally 225 degrees
03F0  03C1              ;KEY 8 - Move straight 180 degrees
  F2  039E              ;KEY 9 - Move diagonally 135 degrees
  F4  0385              ;KEY A - Put shape on hold
  F6  038E              ;KEY B - Restore shape from hold
  F8  045E              ;KEY C - Show current shape table
  FA  046A              ;KEY D - Draw line to cursor
  FC  0489              ;KEY E - Erase last line drawn
  FE  04D5              ;KEY F - Start over/Enter starting X,Y
```

WAIT AND GET HEXPAD KEYPRESS

```
                ;Input:    None
                ;Output:   RF.1=key pressed (waits until key is
                   pressed)
                ;Called by:DOUBLD  Main Loop
                ;Changes:  RF

0400  2F        GETKEY: DEC  RF      ;Value of RF not important, we
  01  8F                GLO  RF      ;  only need to cycle through
                                     ;  all hex digits
  02  52                STR  R2      ;Push RF.0 as comparison with
                                     ;  keyboard
  03  62                OUT  DEV2    ;Output that value (MSD unimportant)
  04  22                DEC  R2      ;R(X) was incremented.  Reset it
  05  E2                SEX  2       ;Kill time to allow keyboard to
  06  E2                SEX  2       ;  respond (i.e. for propagation
                                     ;  delays)
  07  3E 00             BN3  GETKEY  ;If no match, loop.  Key not pressed
  09  7B                SEQ          ;Tone on.  Key pressed
  0A  F8 10       1H:   LDI  $10     ;Set up for loop of about $1000
                                     ;  times
  0C  BF                PHI  RF
  0D  2F          2H:   DEC  RF      ;Loop here to sound the tone at
  0E  9F                GHI  RF      ;  least for a short time period
  0F  3A 0D             BNZ  2B      ;  and for some debounce
  11  36 0A             B3   1B      ;Loop if key still pressed
  13  F0                LDX          ;Pop stack.  LSD=key pressed
  14  FA 0F             ANI  $0F     ;Strip off useless first four bits,
  16  BF                PHI  RF      ;  leaving only the correct digit.
                                     ;  Save in RF.1
  17  7A                REQ          ;Tone off
  18  D5                RETN         ;Return.  Keypress in RF.1
```

CHECK FOR KEY PRESSED

```
                    ;Input:     RF.1=value to test
                    ;Output:    RF.0=0=key not being pressed
                                RF.1=0=key is being pressed
                    ;Called by:Main Loop
                    ;Changes:   RF.0

0419  F8 00    CHEKEY: LDI  $0      ;Preset RF.0=0 as a flag
  1B  AF              PLO  RF
  1C  9F              GHI  RF      ;Get the value to be tested
  1D  52              STR  R2      ;Push to prepare to output
  1E  62              OUT  DEV2    ;Output M(R(X))
  1F  22              DEC  R2      ;Reset stack pointer incremented
                                   ;  by OUT
  20  3E 25           BN3  1F      ;If no match, go to exit
  22  F8 01           LDI  $1      ;Else set RF.0=1 to flag that
  24  AF              PLO  RF      ;  the key is being pressed
  25  D5          1H: RETN         ;Return
```

DISPLAY HEX DIGIT

```
                    ;Input:     RF.1=digit for display
                                RD=destination address in the display page
                    ;Output:    Digit displayed.  RD=RD+1
                    ;Called by:DOUBLD
                    ;Changes:   RD, RE, RF.0

0426  F8 81    DSPHEX: LDI  $81     ;RE.1 addresses an index
  28  BE              PHI  RE      ;  table in VIP ROM containing
  29  9F              GHI  RF      ;  the hex bit patterns for the
  2A  AE              PLO  RE      ;  16 hex digits
  2B  4E              LDA  RE      ;Get start address of bits
  2C  AE              PLO  RE      ;Put in RE.  RE addresses bits now
  2D  F8 05           LDI  $5      ;Set up RF.0 as loop count
  2F  AF              PLO  RF      ;  for 5 bit lines
  30  4E          1H: LDA  RE      ;Get a row of bits from ROM
  31  5D              STR  RD      ;Put in display (destructively)
  32  8D              GLO  RD      ;Add 8 to RD using double
  33  FC 08           ADI  $8      ;  precision arithmetic to
  35  AD              PLO  RD      ;  move RD down one bit
  36  9D              GHI  RD      ;  row in the display
  37  7C 00           ADCI $0
  39  BD              PHI  RD
  3A  2F              DEC  RF      ;Count number lines shown
  3B  8F              GLO  RF      ;Test the counter
  3C  3A 30           BNZ  1B      ;If≠0, loop to do another bit row
```

```
043E  8D              GLO  RD     ;Reset RD so that it is
 3F  FF 27            SMI  $27    ;  exactly one byte past
 41  AD               PLO  RD     ;  where it was on entry
 42  9D               GHI  RD     ;  Thus subsequent calls
 43  7F 00            SMBI $0     ;  may more easily display
 45  BD               PHI  RD     ;  a string of digits
 46  D5               RETN         ;Return from subroutine
```

GET AND DISPLAY DOUBLE DIGIT HEX VALUE

```
                ;Input:    RD=destination address for display
                ;Output:   RF.1=double digit hex value also displayed
                           RD=RD+2
                ;Called by:INIT
                ;Calls:    GETKEY  DSPDIG
                ;Changes:  RF.1

0447  D4 04 00 DOUBLD: CALL GETKEY ;Get the first digit
 4A  D4 04 26          CALL DSPDIG ;Display @ RD (RD=RD+1)
 4D  9F               GHI  RF     ;Get digit passed back in RF.1
 4E  FE               SHL         ;Shift it left four times
 4F  FE               SHL         ;  to move the digit over
 50  FE               SHL         ;  to the most significant
 51  FE               SHL         ;  digit position
 52  73               STXD        ;Push onto stack
 53  D4 04 00         CALL GETKEY ;Get the second digit
 56  D4 04 26         CALL DSPDIG ;And display it @ RD (RD=RD+1)
 59  60               IRX         ;Point to saved digit on stack
 5A  9F               GHI  RF     ;Get new second digit
 5B  F1               OR          ;Combine via logical OR
 5C  BF               PHI  RF     ;Place in RF.1 to pass back
 5D  D5               RETN        ;Return
```

KEY C - DRAW CURRENT SHAPE

```
                ;Input:    A shape table at $0700 ending with FFFF
                   stop.
                           R7.1=page address of shape (usually=$07)
                ;Output:   Shape drawn.  R7 addresses the FFFF stop
                   indicator
                ;Called by:Main Loop  RECVR
                ;Calls:    CLEAR  SHAPE

045E  D4 01 30   KEYC: CALL CLEAR  ;Erase display
 61  F8 00            LDI  $00    ;Set R7= $0700 the address
 63  A7               PLO  R7     ;  of the shape table
 64  D4 00 84         CALL SHAPE  ;Draw the shape
 67  27               DEC  R7     ;R7 addresses the FFFF
 68  27               DEC  R7     ;  stop indicator again
 69  D5               RETN        ;Return
```

KEY D - DRAW LINE

```
                ;Input:    RC=cursor X,Y
                ;Output:   Line drawn to cursor (unless Shape table
                             is full)
                           X,Y and $FFFF placed in Shape table
                ;Called by:Main Loop
                ;Calls:    LINE
                ;Changes:  R7, RD

046A  87          KEYD: GLO  R7     ;Test if R7 is at end of
 6B  FB FE              XRI  $FE    ;  shape table
 6D  32 88              BZ   1F     ;If so, then go to exit
 6F  9C                 GHI  RC     ;Set RD=RC which will
 70  BD                 PHI  RD     ;  be the "to" point (X1,Y1)
 71  8C                 GLO  RC     ;  for the line to be
 72  AD                 PLO  RD     ;  drawn
 73  27                 DEC  R7     ;Point R7 to the last (X0,Y0)
 74  27                 DEC  R7     ;  coordinates in the shape table
 75  47                 LDA  R7     ;Transfer X0,Y0 to RC which will
 76  AC                 PLO  RC     ;  be the "from" point
 77  47                 LDA  R7     ;  of the line (R7 is left at
 78  BC                 PHI  RC     ;  its previous value)
 79  D4 01 47           CALL LINE   ;Draw the line from RC to RD
 7C  8C                 GLO  RC     ;Transfer RC to the shape table
 7D  57                 STR  R7     ;  increasing it by two bytes.
 7E  17                 INC  R7     ;  RC is equal to X1,Y1 because
 7F  9C                 GHI  RC     ;  of LINE
 80  57                 STR  R7
 81  17                 INC  R7
 82  F8 FF              LDI  $FF    ;Then insert an $FFFF stop
 84  57                 STR  R7     ;  indicator into the last
 85  17                 INC  R7     ;  two bytes of the
 86  57                 STR  R7     ;  table.
 87  27                 DEC  R7
 88  D5                 RETN        ;Return from subroutine
```

KEY E - ERASE LINE

```
;Input:    None.  Usually a shape table with R7
            addressing some X,Y
;Output:   Last line erased.  X1,Y1 overwritten
            by $FFFF
;Called by:Main Loop
;Calls:    KEYC
;Changes:  R7, RC, RD
```

```
0489  87          KEYE: GLO  R7     ;Test if R7.0≤2 meaning the
 8A  FF 03             SMI  $3      ;  shape table is empty
 8C  3B A1             BM   1F      ;If so, cancel routine - no
                                    ;  lines to erase
 8E  FF 01             SMI  $1      ;Else set R7=R7-4 to address
 90  A7                PLO  R7      ;  X0,Y0 in table
 91  E7                SEX  R7      ;X=7 for upcoming transfers
 92  72                LDXA
 93  AC                PLO  RC      ;RC.0=X0
 94  72                LDXA
 95  BC                PHI  RC      ;RC.1=Y0
 96  72                LDXA
 97  AD                PLO  RD      ;RD.0=X1
 98  F0                LDX
 99  BD                PHI  RD      ;RD.0=Y1
 9A  F8 FF             LDI  $FF     ;Set FFFF stop indicator
 9C  73                STXD         ;  into shape table, erasing
 9D  57                STR  R7      ;  the last X1,Y1 points
 9E  D4 04 5E          CALL KEYC    ;Show shape to this point
 A1  D5           1H:  RETN         ;Return
```

04A2 - 04A6 -- Available bytes. Don't ask.

OR/XOR SUB

```
;Input:    None
;Output:   $012D in POINT sub flopped from OR to XOR
            or from XOR to OR
;Called by:CURSOR
;Changes:  RF and M($012D)
;Warning:  First instructions in a program should
            initialize $012D to a known starting value
            (either $F1 for OR or $F3 for XOR). Ignoring
            this warning could lead to odd intermittent
            bugs upon using the reset switch.

04A7  F8 01      ORXOR: LDI  $01     ;Set RF= address of the OR
 A9  BF                 PHI  RF      ;  or XOR instruction
 AA  F8 2D              LDI  $2D     ;  in POINT sub. (@$012D)
 AC  AF                 PLO  RF
 AD  0F                 LDN  RF      ;Load current instruction
 AE  FB 02              XRI  $2      ;Change OR to XOR/XOR to OR
 B0  5F                 STR  RF      ;Put new instruction back
 B1  D5                 RETN         ;Return from subroutine
```

*Note: This sub changes a byte inside POINT. If POINT is put into ROM, such a "self modification" would not work.

SHOW AND ERASE CURSOR @ RC

```
                ;Input:    RC is cursor X,Y
                ;Output:   Cursor blinked once
                ;Called by:Main Loop
                ;Calls:    TIMER  ORXOR
                ;Changes:  RE, RF (in TIMER sub)

04B2  D4 04 A7 CURSOR: CALL ORXOR  ;Change POINT to XOR
  B5  9C               GHI  RC     ;Transfer cursor X,Y to
  B6  BE               PHI  RE     ;  RE for displaying
  B7  8C               GLO  RC
  B8  AE               PLO  RE
  B9  D4 01 08         CALL POINT  ;Do sub -- show cursor
  BC  D4 04 CD         CALL TIMER  ;Wait for a little
  BF  9C               GHI  RC     ;Again, transfer RC to
04C0  BE               PHI  RE     ;  RE this time to
  C1  8C               GLO  RC     ;  erase the cursor
  C2  AE               PLO  RE
  C3  D4 01 08         CALL POINT  ;Do sub -- erase cursor
  C6  D4 04 CD         CALL TIMER  ;Wait -- causes blink
  C9  D4 04 A7         CALL ORXOR  ;Reset POINT to OR
  CC  D5               RETN        ;Return from subroutine

04CD  F8 04     TIMER: LDI  $4     ;Set RF.1=timer loop=
  CF  BF               PHI  RF     ;  speed of cursor
  D0  2F             1H: DEC  RF   ;Decrement RF
  D1  9F               GHI  RF     ;Get high part of RF
  D2  3A D0            BNZ  1B     ;If not zero yet, loop back
  D4  D5               RETN        ;Return.  RF.1=00/RF.0=FF
```

INITIALIZE/RESET ON KEY F

```
                ;Input:    R7.1=page of picture buffer
                ;Output:   Buffer set to $FF bytes
                           User inputs X,Y into RC and into X0,Y0
                  in shape table
                           Screen cleared
                ;Called by:Main Loop  HOLD
                ;Calls:    CLEAR  DOUBLD
                ;Changes:  R7.0, RC, RD

04D5  D4 01 30   INIT: CALL CLEAR  ;Erase display
  D8  E7               SEX  R7     ;X=7
  D9  F8 FF            LDI  $FF
  DB  A7               PLO  R7     ;Set entire shape table
```

```
04DC  F8 FF        1H: LDI  $FF     ;  to $FF bytes so any
 DE   73               STXD         ;  size shape table will
 DF   87               GLO  R7      ;  work.  Note that last byte
 E0   3A DC            BNZ  1B      ;  (i.e. first in table) is
                                    ;  not set
 E2   AD               PLO  RD      ;RD.0=00 (D register is zero here)
 E3   9B               GHI  RB      ;RD=top left corner of display
 E4   BD               PHI  RD      ;  for showing hex digits
 E5   D4 04 47         CALL DOUBLD  ;Get X coordinate
 E8   9F               GHI  RF
 E9   FA 3F            ANI  $3F     ;Limit X to 00-3F range
 EB   AC               PLO  RC      ;  and put in RC.0
 EC   57               STR  R7      ;Also put into shape table
 ED   17               INC  R7      ;  via R7
 EE   D4 04 47         CALL DOUBLD  ;Get Y coordinate
 F1   9F               GHI  RF
 F2   FA 7F            ANI  $7F     ;Limit to 00-7F range
 F4   BC               PHI  RC      ;  and put in RC.1
 F5   57               STR  R7      ;Also put into shape table
 F6   17               INC  R7      ;  advancing R7.0 to $02
 F7   D4 01 30         CALL CLEAR   ;Erase numbers from screen
 FA   D5               RETN         ;Return

                ;End SHAPE MAKER
```